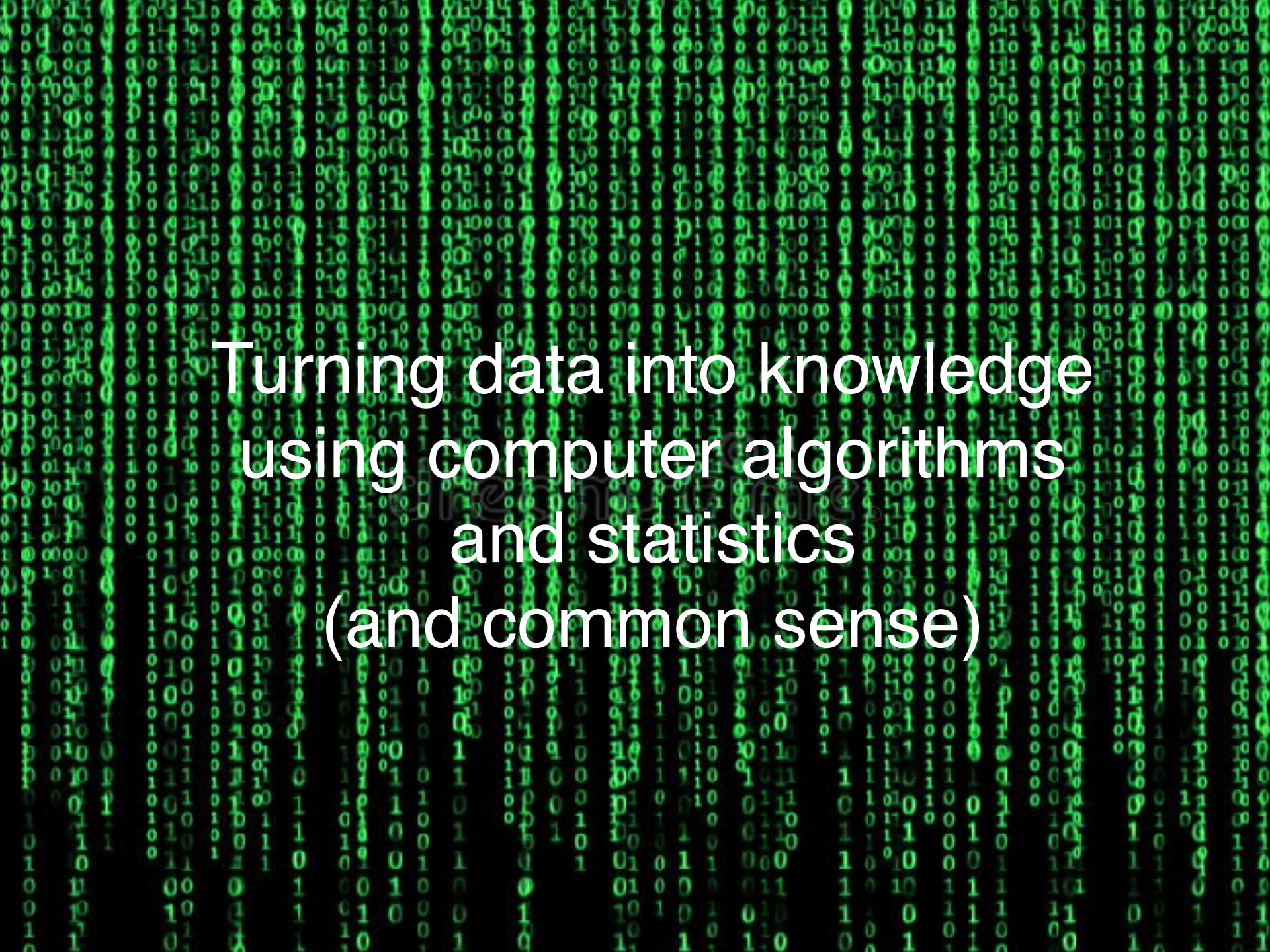
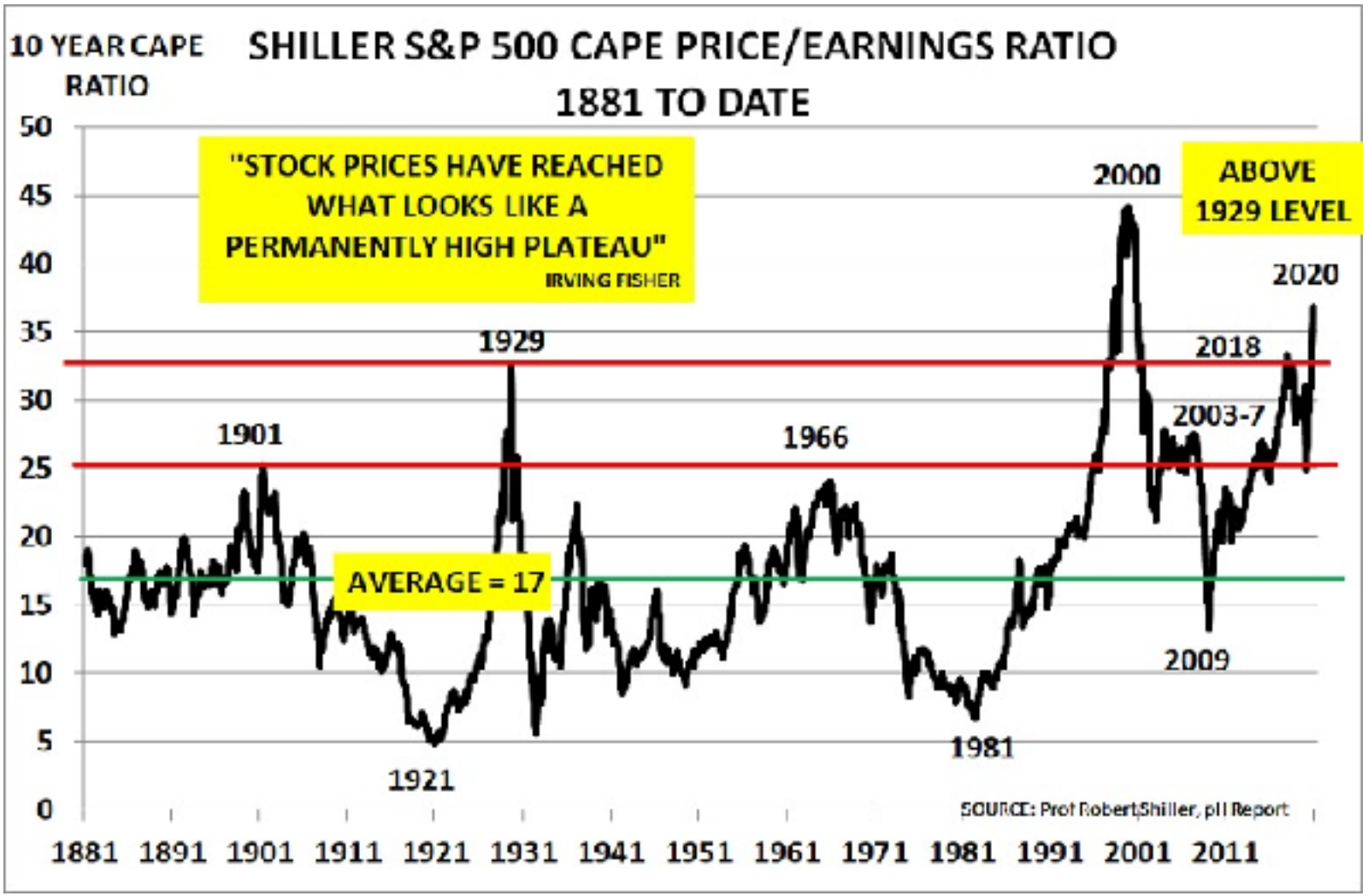
Data Analysis and Machine Learning using Python

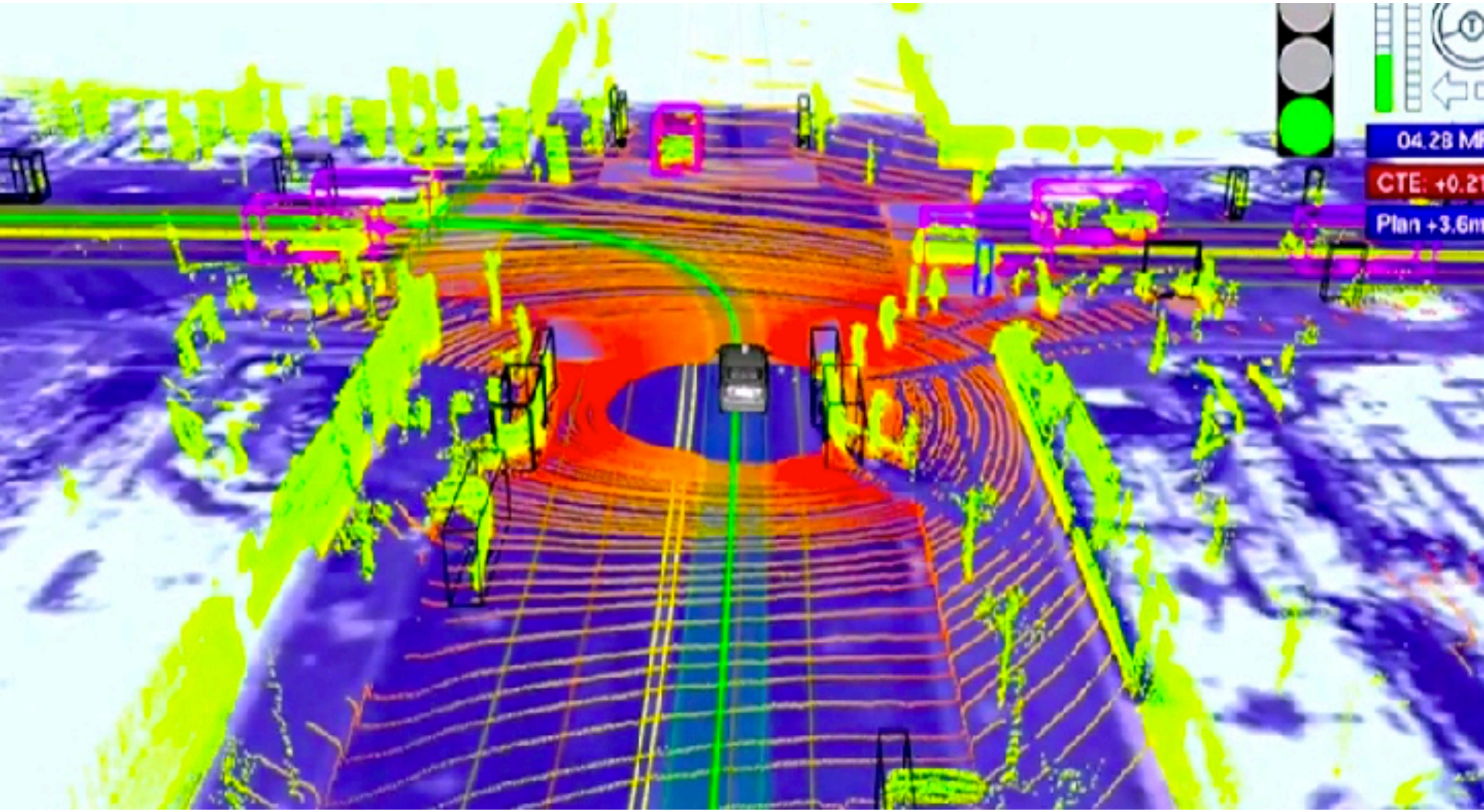
Lecture 1: *Course overview and introduction*
March 9 2024

Gunther Roland
Professor of Physics

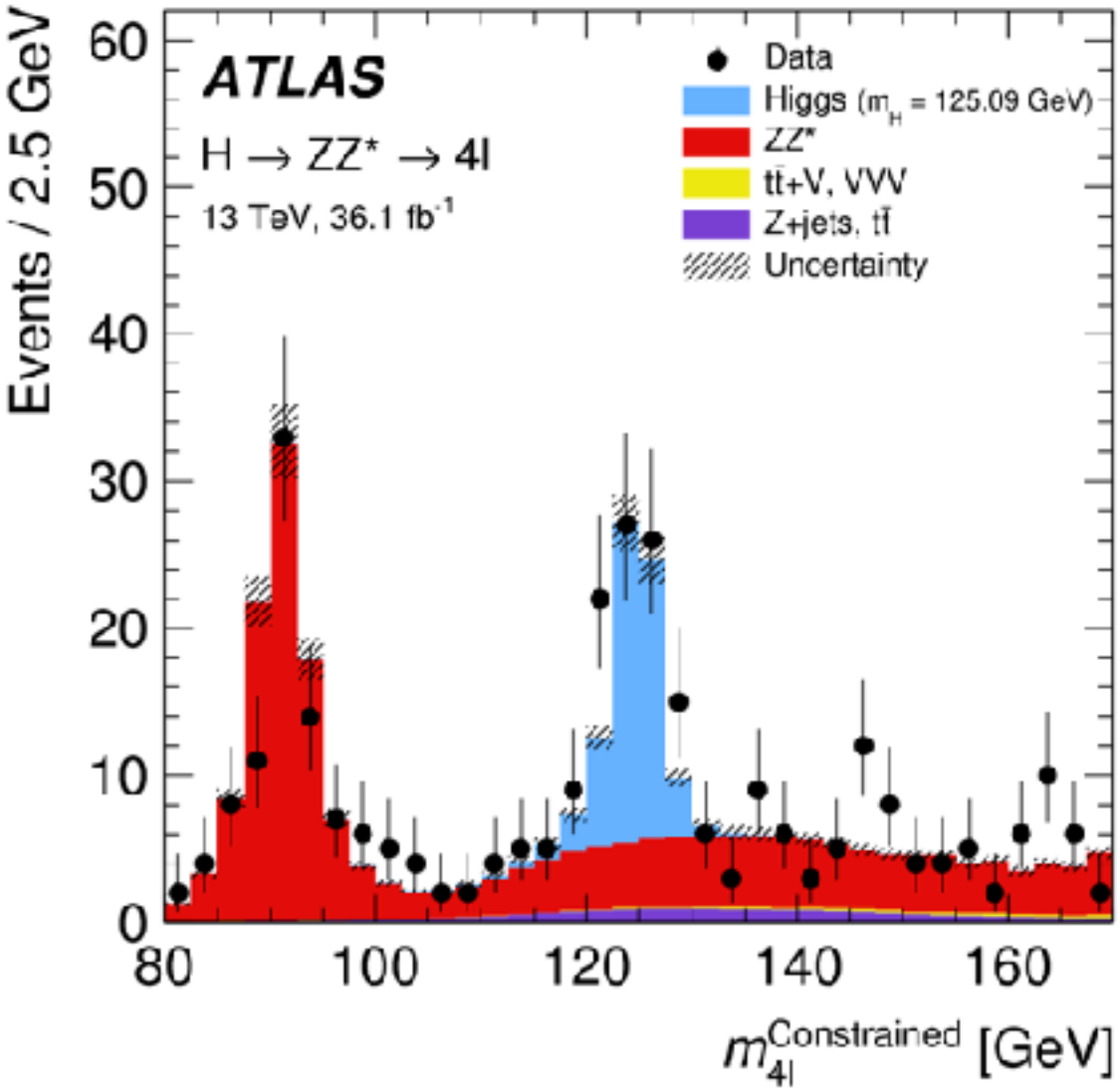

Turning data into knowledge
using computer algorithms
and statistics
(and common sense)



Describe patterns and correlations in financial markets
(and, maybe, predict future)



Make a dynamic model of surroundings
(and, definitely, predict where things will move to)



Discover new laws of nature
(and, possibly, predict other new phenomena)

Me

Prof. Gunther Roland

gunthermroland@gmail.com

Skype: Phobosrolandg

Slack

Experimental physicist

Working at CERN laboratory in Geneva, Switzerland (CMS experiment at LHC accelerator) and BNL laboratory, Long Island, USA (sPHENIX experiment at RHIC accelerator)

Big Data in Physics

- CMS experiment studying subatomic particle collisions
- Studying several billion particle collisions each second
- Produces several Terabyte of data each second → 100 Exabytes/year
  - your laptop probably has ~Terabyte hard drive; this would be 100 million of such hard drives to store
- We are looking for **few** “special” collisions in 100.000.000.000.000.000 to find new laws of nature

What do I want you to learn?

- 10 class sessions
- First 5 sessions: Introduction to Python and basic concepts of data analysis

No.	Topics
Lecture 1	**Introduction to Programming in Python** ● Variables, Types, Functions ● Branching, Iteration, Strings, Lists
Lecture 2	**Data Analysis and Basic Statistics** ● Scientific Approach to Data Analysis and Basic Statistics ● Statistical and Systematic Uncertainties
Lecture 3	**Essential Python Libraries for Data analysis** ● Using Numpy and Scipy ● Plotting using Matplotlib
Lecture 4	**Data Visualization and Working with Large Datasets** ● Principles of Data Visualization ● Best Practices in Working with Large Data Sets ● Working with Pandas
Lecture 5	**Introduction to Multivariate Analysis** ● Introduction to the Concept of Multivariate Analysis ● Applications and Significance in Data Analysis

What do I want you to learn?

- Second 5 sessions: Introduction to Machine learning and applications of various ML techniques

Lecture 6	**Introduction to Machine Learning** ● What is Machine Learning? ● Basic Classifications in Machine Learning
Lecture 7	**Supervised Learning in Scikit-learn** ● Overview and use of Scikit-learn ● Decision Tree Algorithm ● Multi-layer Perceptrons
Lecture 8	**Unsupervised Learning** ● K-Means Clustering Algorithm ● Principal Component Analysis (PCA)
Lecture 9	**Reinforcement Learning** ● Concepts and Basic Elements of Reinforcement Learning ● Markov Decision Processes
Lecture 10	**Deep Learning** ● Overview of Deep Learning ● Convolutional Neural Networks (CNN) ● Recurrent Neural Networks (RNN) ● Generative Adversarial Networks (GAN)

Some more thoughts

- Programming background
  - Focus on understanding concepts
  - Focus on essentials, not elegance or ~~**complexity**~~
  - Not going for the most fancy/short/clever code
- Tools
  - Laptop w/ Python v3.5 or higher
  - Will give examples as Jupyter notebooks

TA sessions

- Please make use of the TA sessions to ask questions and practice
- Also, use the chat feature during the lecture to ask questions - the TA will reply right away or let me know so that I can answer questions in the next lecture
- Please let your TA know:
  - What is your programming background?
  - Have you used machine learning techniques before?
  - Do you have Python installed on your laptop?
    - suggest using Jupyter or Spyder
    - Make sure you have Python running for homework

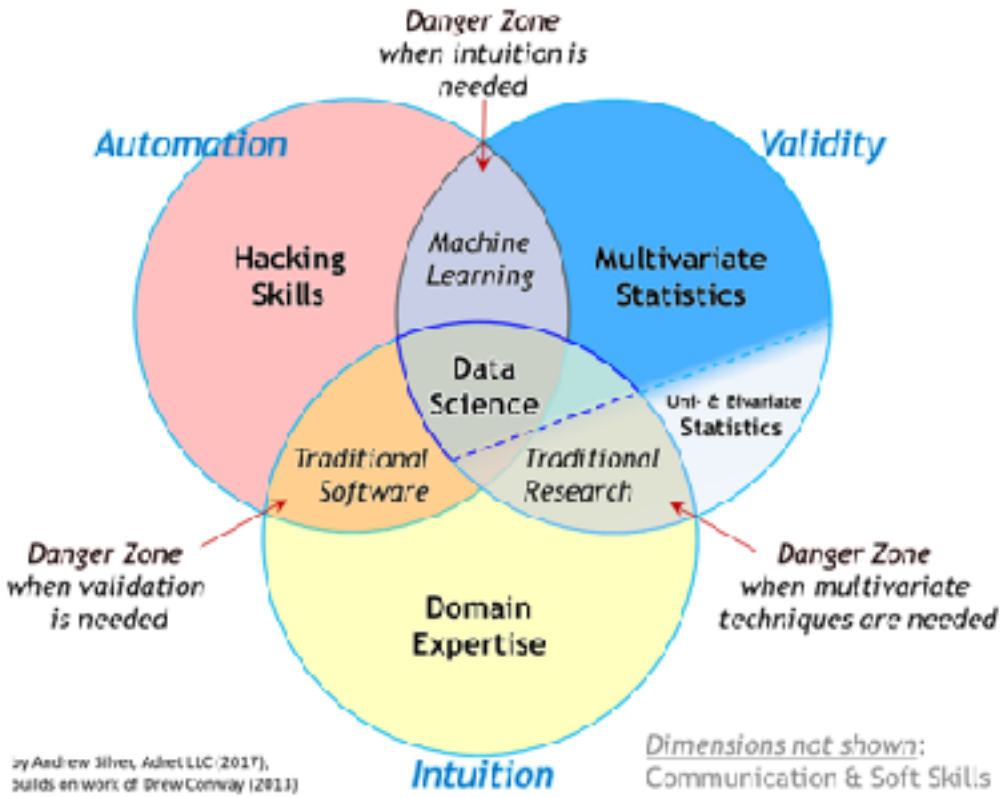
Learning

- How many times do you have to watch this clip until you can make the same forehand shot?

Learning by doing

- How many times do you have to watch this clip until you can make the same forehand shot?
- That's not how it works for tennis, and it's the same for any challenging activity - you learn by doing!
- Practice, practice, practice….that's what we are going to do

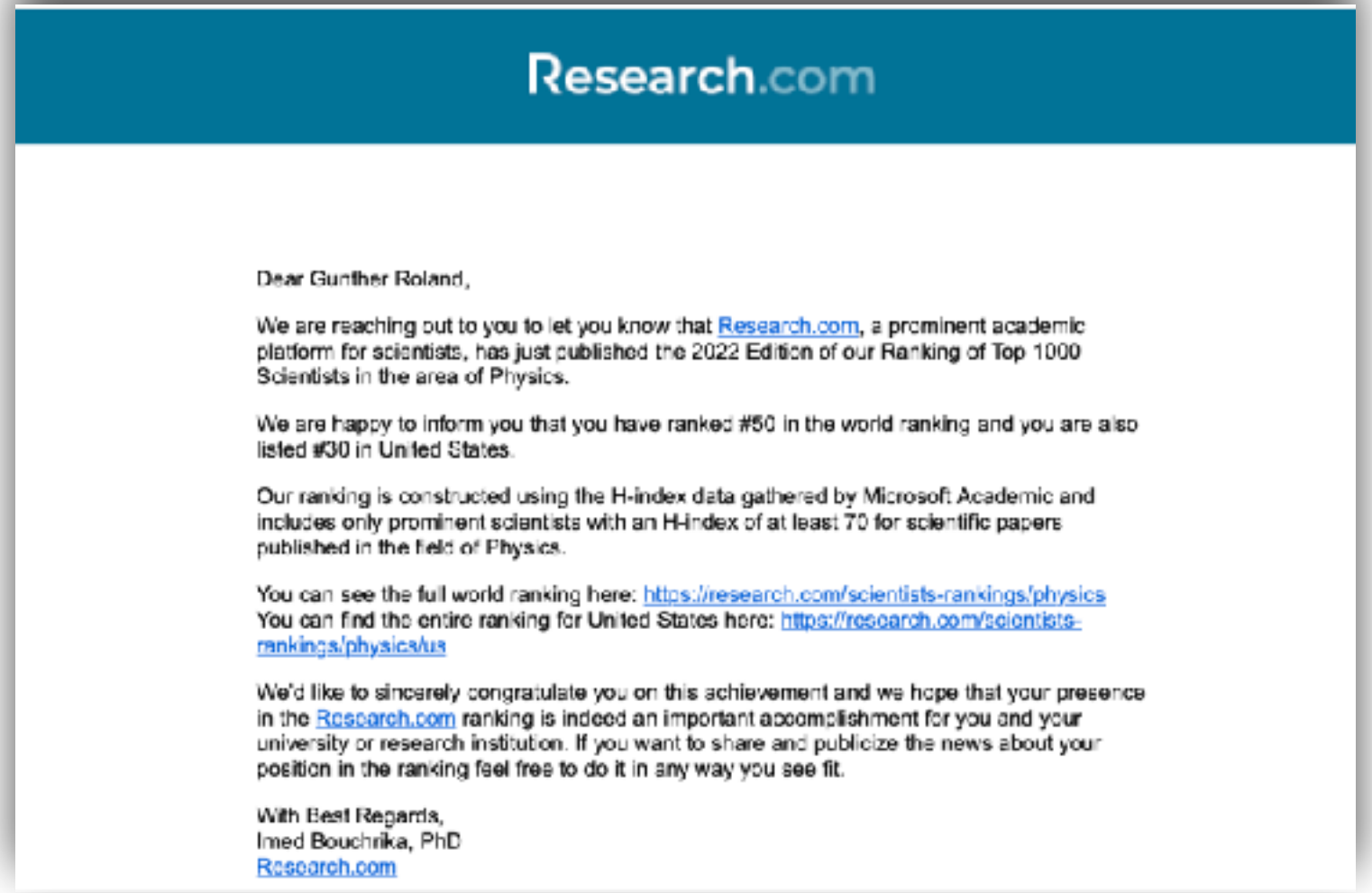
What is Data Science?



wikipedia

Data science is an interdisciplinary field that uses scientific methods, processes, algorithms and systems to extract knowledge and insights from noisy, structured and unstructured data, and apply knowledge from data across a broad range of application domains.

Data science gone wrong



Research.com

Dear Gunther Roland,

We are reaching out to you to let you know that Research.com, a prominent academic platform for scientists, has just published the 2022 Edition of our Ranking of Top 1000 Scientists in the area of Physics.

We are happy to inform you that you have ranked #50 in the world ranking and you are also listed #30 in United States.

Our ranking is constructed using the H-index data gathered by Microsoft Academic and includes only prominent scientists with an H-index of at least 70 for scientific papers published in the field of Physics.

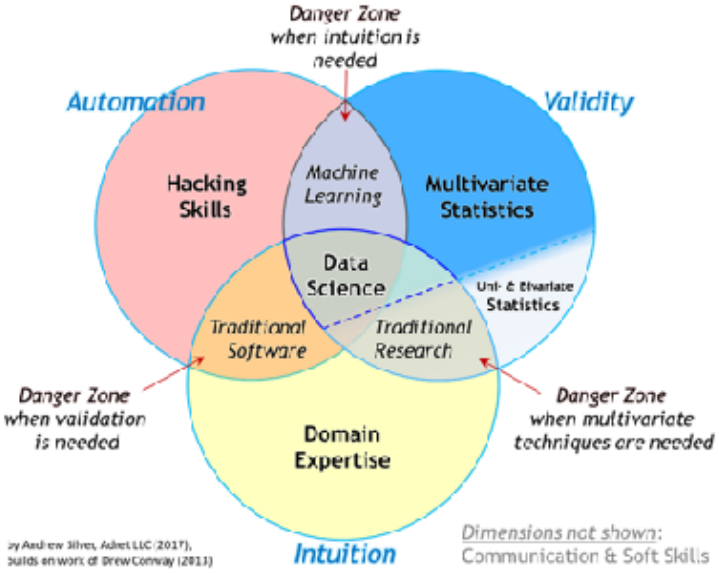
You can see the full world ranking here: https://research.com/scientists-rankings/physics
You can find the entire ranking for United States here: https://research.com/scientists-rankings/physics/us

We'd like to sincerely congratulate you on this achievement and we hope that your presence in the Research.com ranking is indeed an important accomplishment for you and your university or research institution. If you want to share and publicize the news about your position in the ranking feel free to do it in any way you see fit.

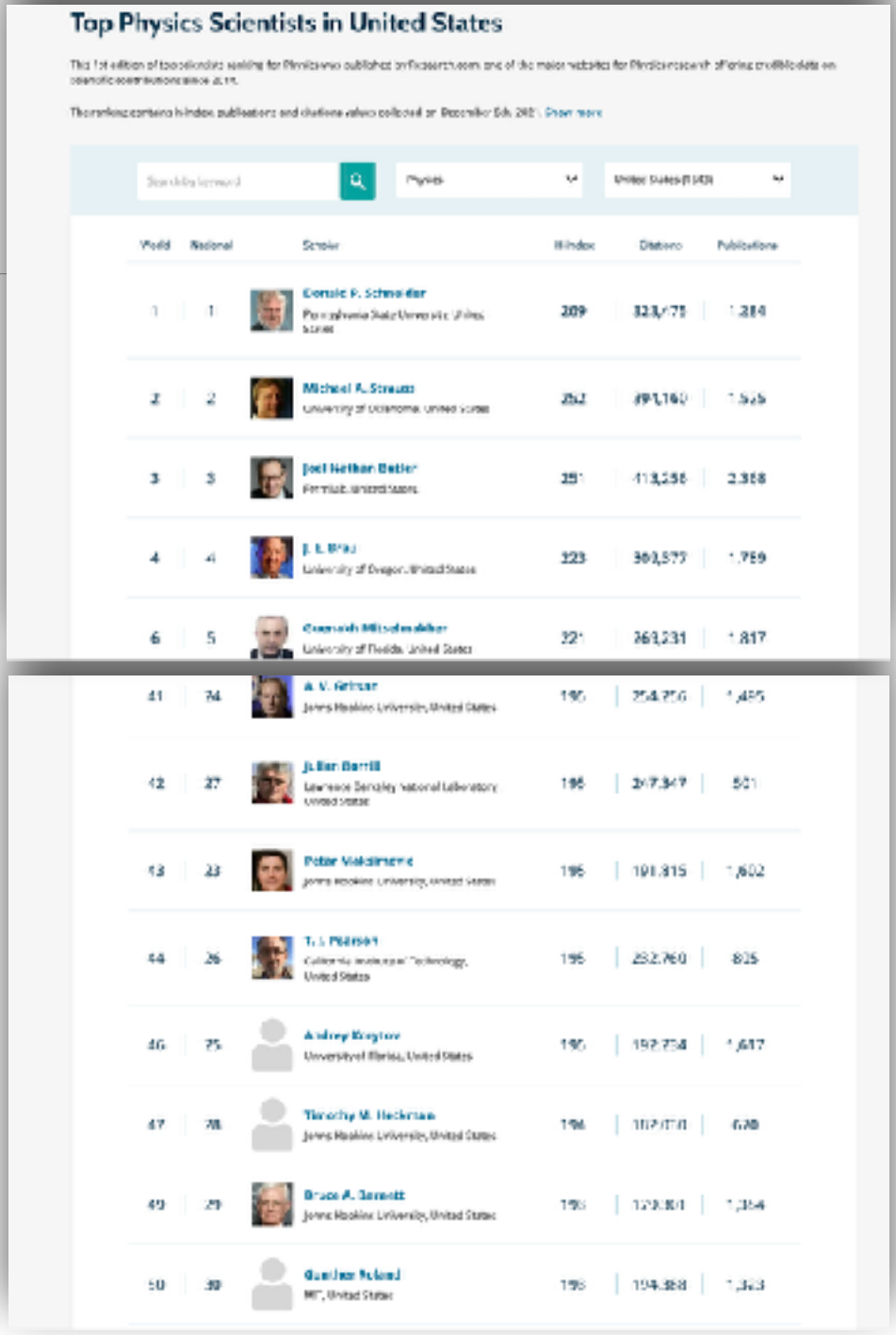
With Best Regards,
Imed Bouchrika, PhD
Research.com

email from research.com earlier this week…

Data science gone wrong



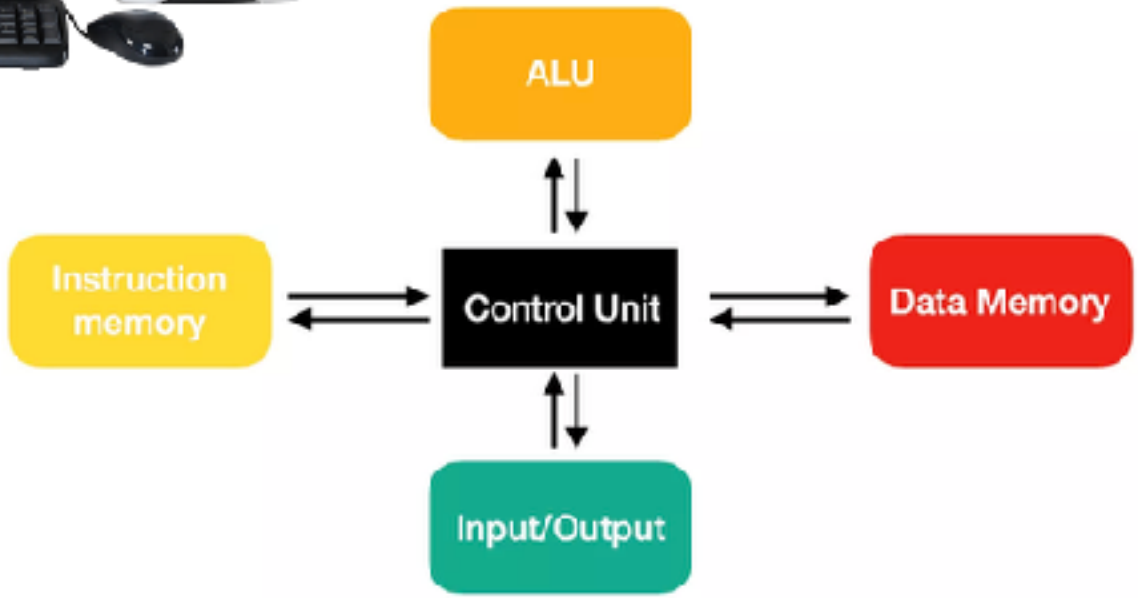
- The “hacking” worked
- The “statistics” are correct
- The conclusions are nonsense
- - did not employ “Domain Expertise”



Goals

- Acquire/practice skills for using Python to understand data
- Gain understanding of statistical analysis - what can data tell you, and what is noise?
- Experience with predictive statistics (modeling) and multivariate analysis, including machine learning

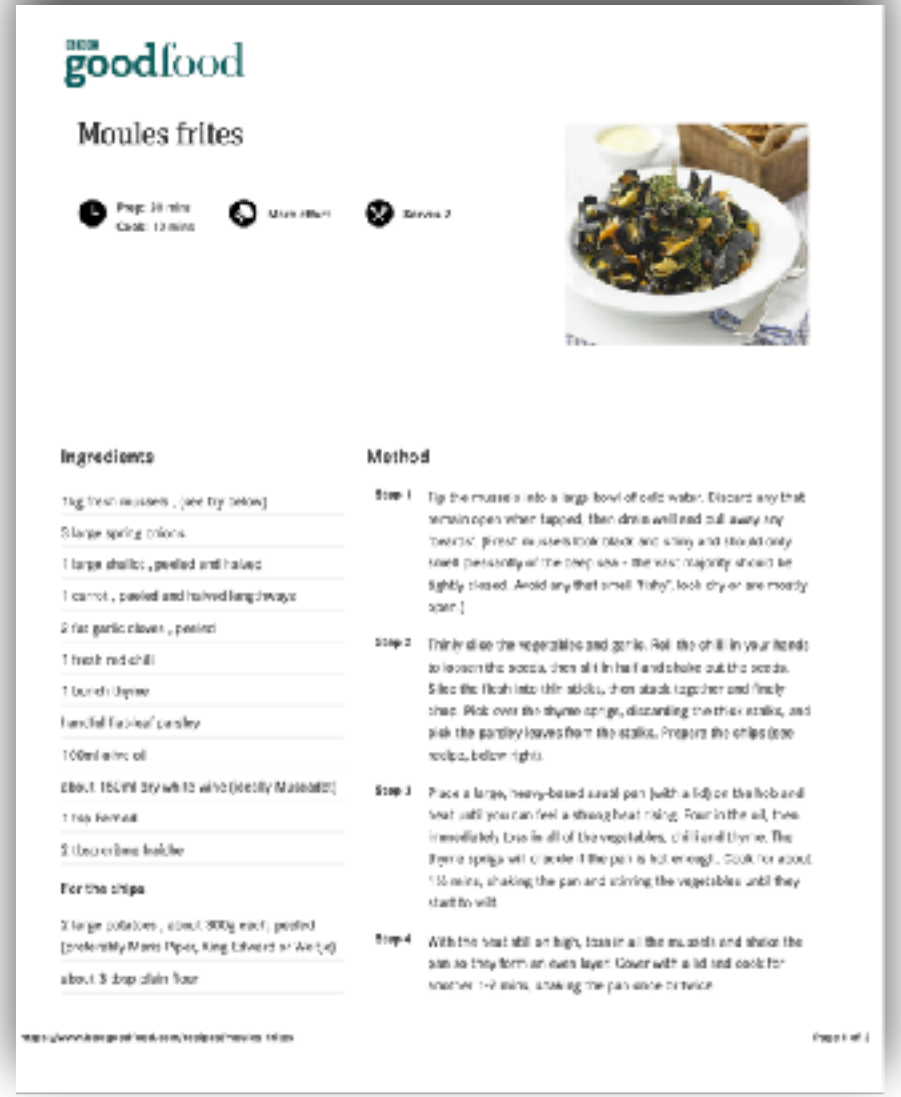
What's a computer?



What does it do for us?

- Allows access to data and storage of data
  - 100's of Gigabytes on your laptop
- Performs calculations
  - Billions or trillions of calculations per second
  - Apple M1 CPU allows up to 2.8 TFlops (TFop = 10^{12} floating point operations per second)
- Built-in calculations (operating system/programming language)
- Complex operations defined by you, the programmer!

Algorithms and programs: think recipe



BBC goodfood

Moules frites

Ingredients

Method

- Sequence of steps
- Flow control determining order of execution
- A means to stop execution (food is done)
- → Algorithm!
- Programming:
  - translate algorithm into code computer can execute
  - Using specific programming language (e.g., Python)

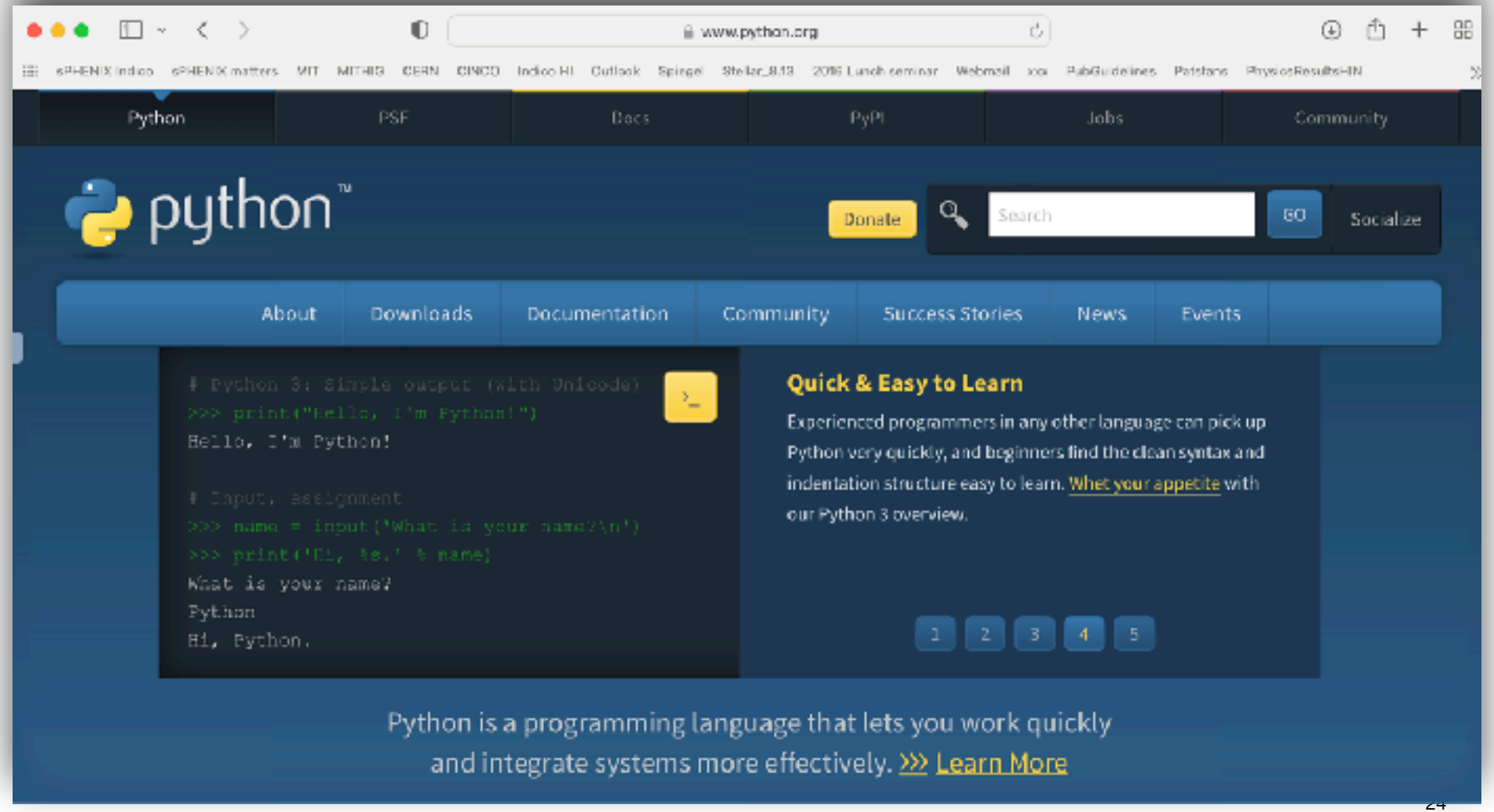
What is a program?

- Sequence of instructions the computer can execute
- Built from **primitive instructions**, e.g.
  - arithmetic and logic operations
  - simple tests, e.g., for flow control
  - moving data (e.g., input and output, copying data)
- also built from predefined sets of instructions (functions), often provided as **libraries**
- Includes flow control to change sequence depending on input, results and to stop program when done
- Executed (translated into computer's machine code) by special program, i.e, Interpreter or Compiler → **programming language**

Example of an algorithm

- Calculate square root of x, i.e., y such that y*y = x
- Algorithm
  1. guess number y
  2. if y*y close enough to x: we're done
  3. otherwise average y and x/y to replace y
  4. check if y*y is close enough to x; otherwise repeat step 3
- Later we'll try to turn this algorithm into a Python program

Our programming language: Python



Python

Python is a high-level, interpreted, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation.[31]

Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.[32][33]

We'll discuss what these words mean

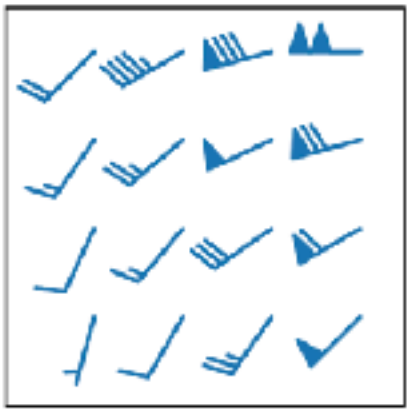
Wikipedia

- Invented in the Netherlands by Guido van Rossum (early 90's)
- Named after Monty **Python**
- Initially often thought of as *scripting language*
  - now used for much more
- Open source
- Extensive selection and use of open-source *libraries*
- One of the most popular programming languages

Visualizing data with plots is essential: Matplotlib

matplotlib.org

matplotlib Plot types Examples Tutorials Reference Usage guide Develop Release notes



barbs(X, Y, U, V)

Matplotlib: Visualization with Python

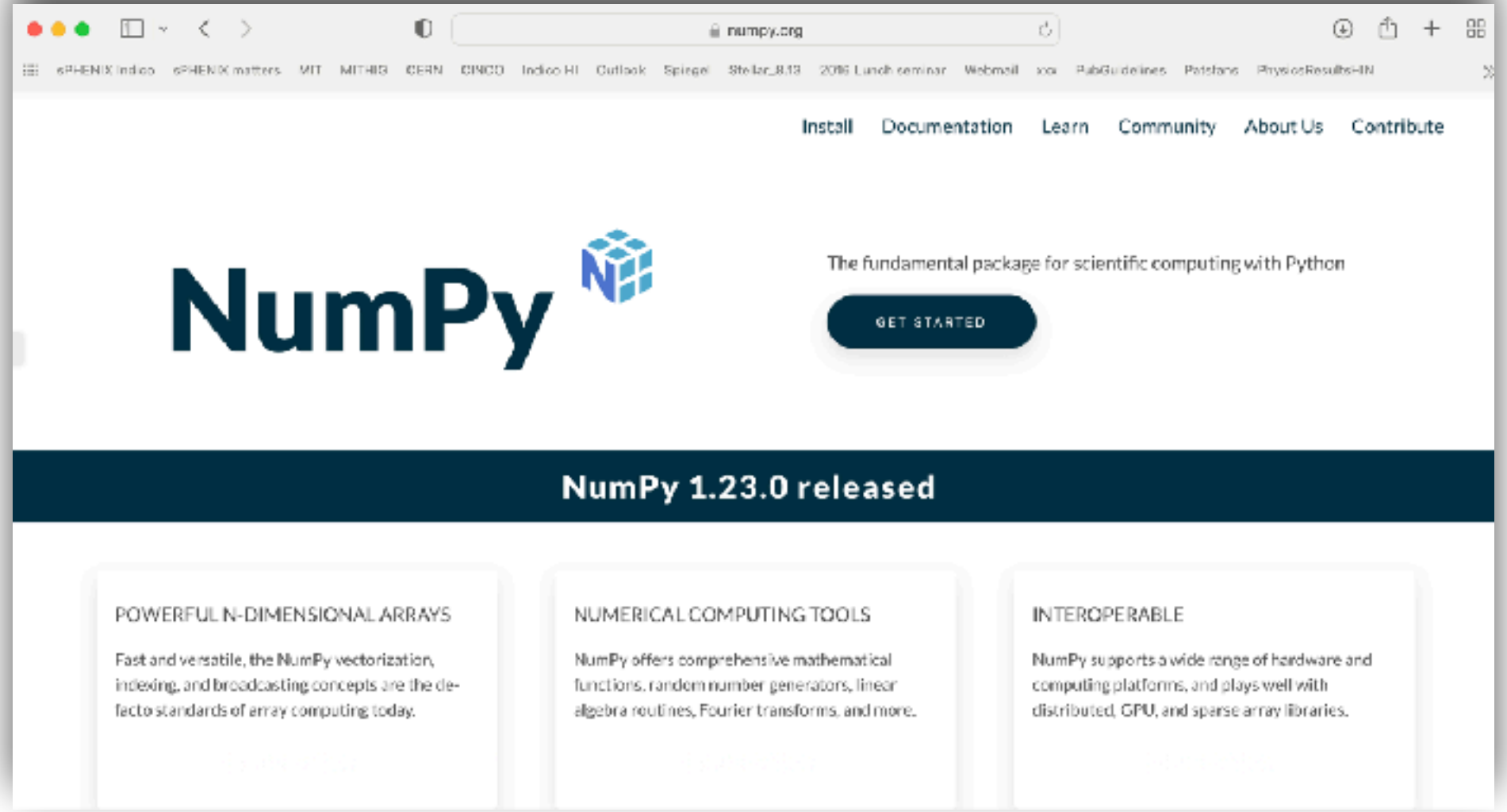
Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible.

- Create publication quality plots.
- Make interactive figures that can zoom, pan, update.
- Customize visual style and layout.
- Export to many file formats .
- Embed in JupyterLab and Graphical User Interfaces.
- Use a rich array of third-party packages built on Matplotlib.

Try Matplotlib (on Binder) →

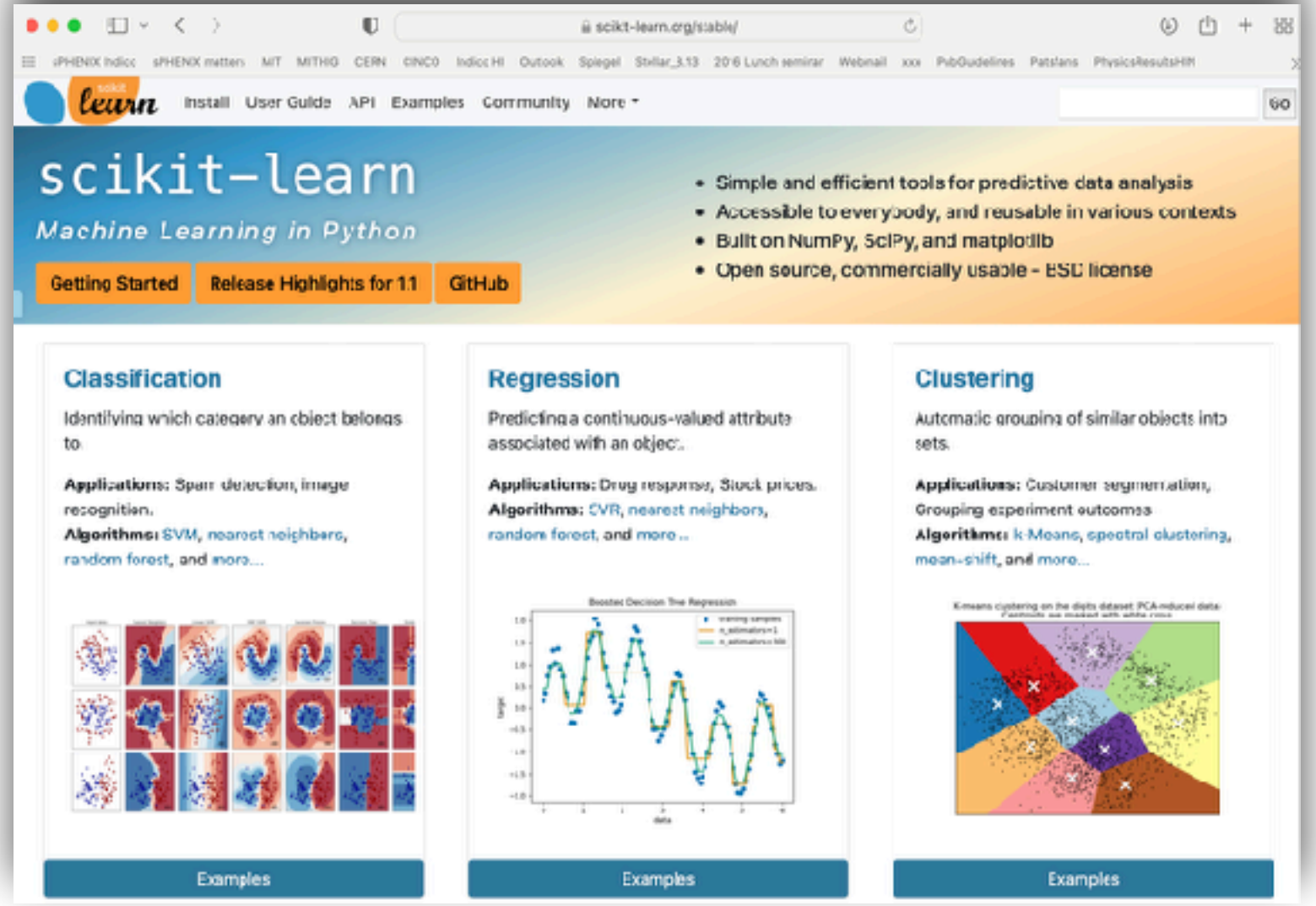
Getting Started Examples Reference Cheat Sheets Documentation

Need a toolbox of numerical algorithms: NumPy



and a math and science toolkit: SciPy

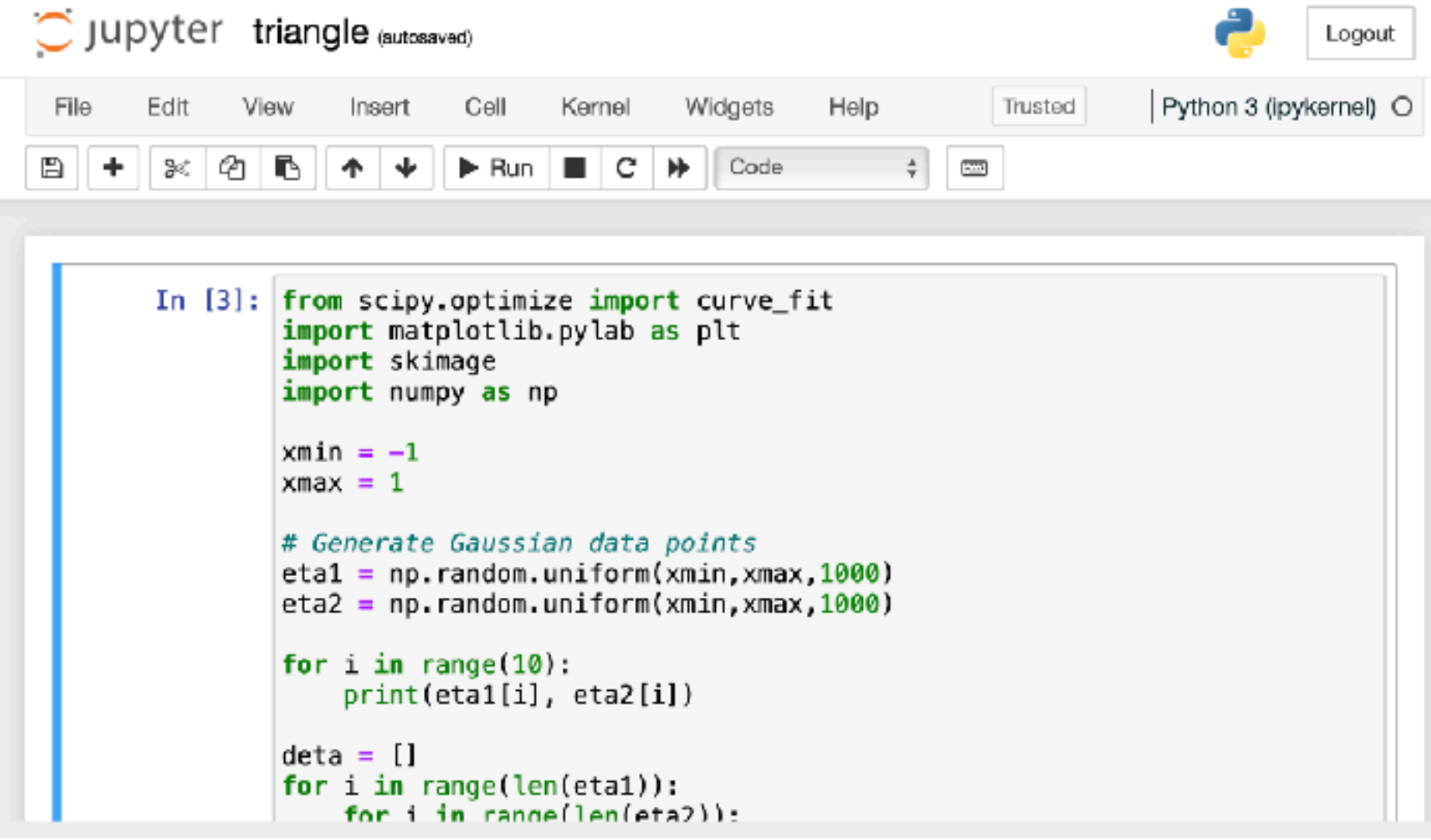
and for machine learning: scikit-learn



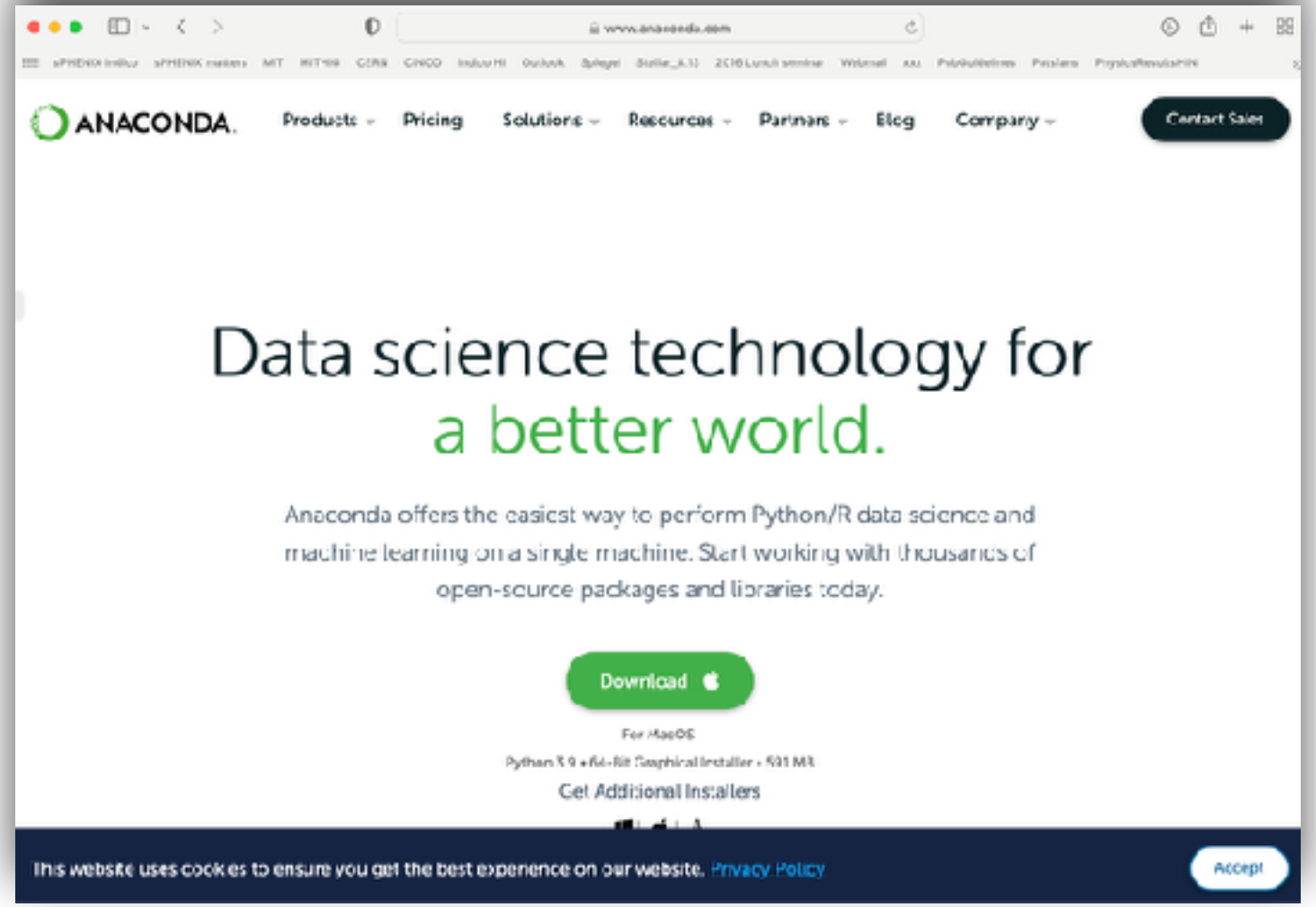
Running Python

- Which computer/operating system are you using (Windows, Mac OSX, Linux?
- Do you have Python installed?
- Recommend use of an IDE (e.g., spyder, Jupyter, etc)

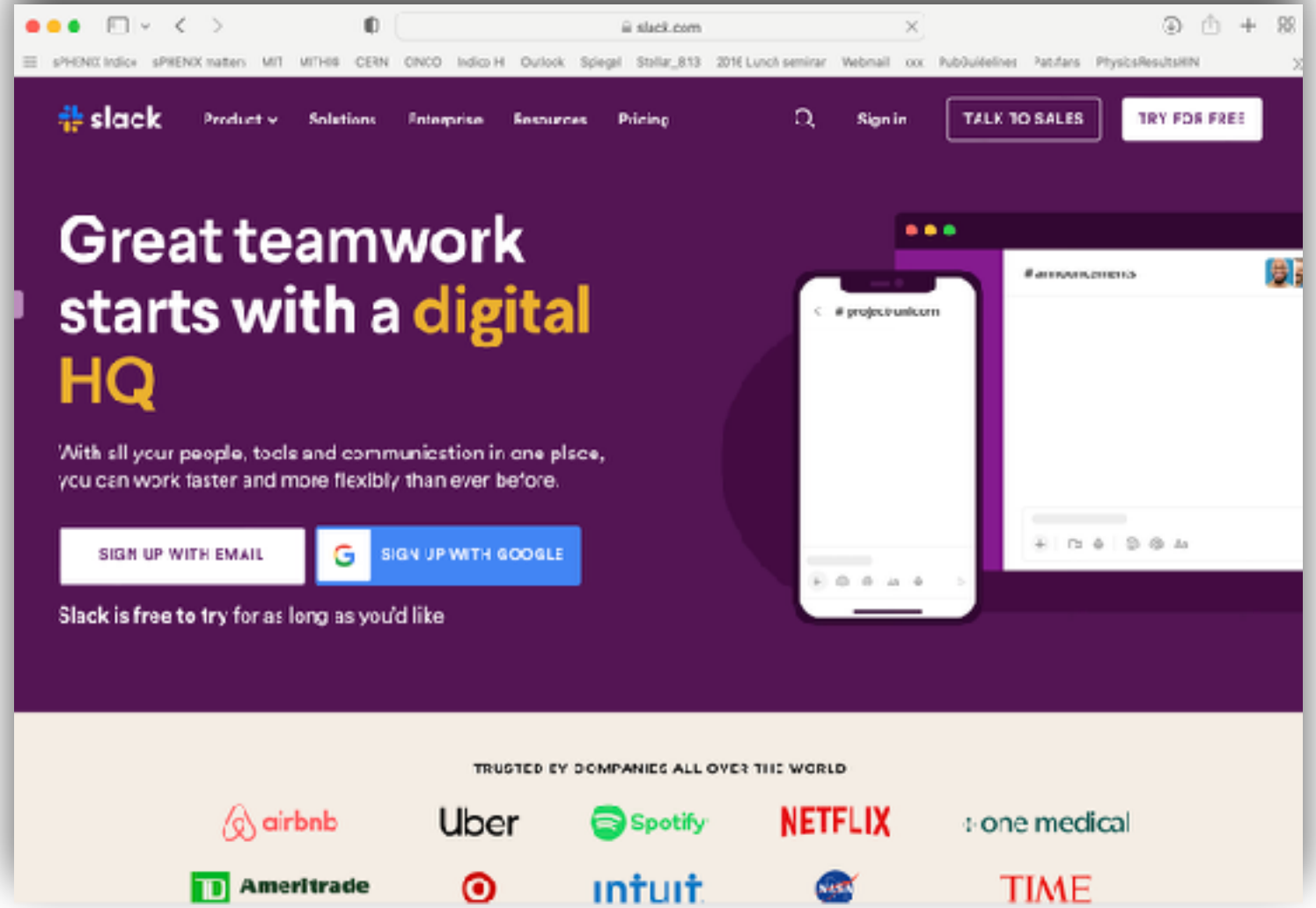
Proposed programming environment (IDE): jupyter



For installing jupyter, I use ANACONDA



and we'll communicate over Slack



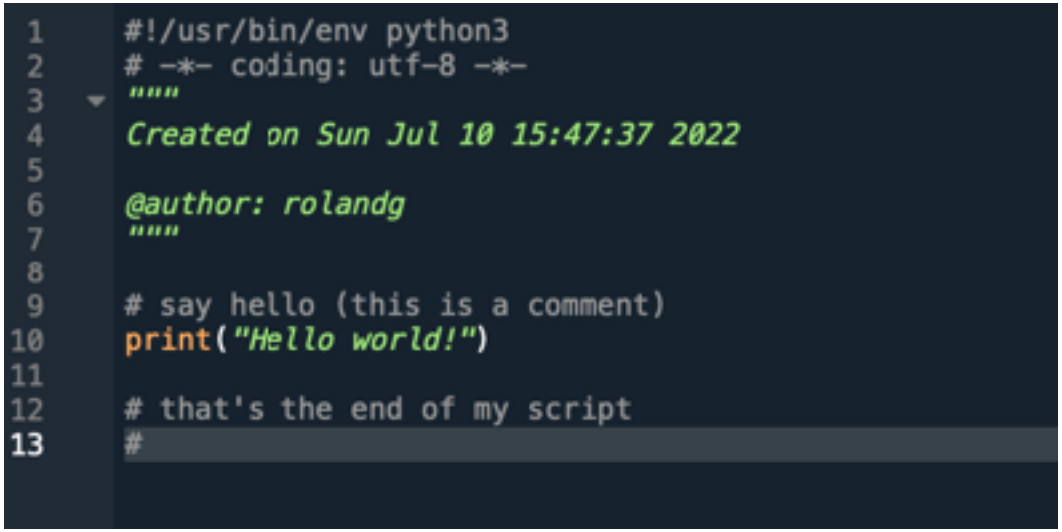
Slack

- How many of you have an account on Slack already?
- Let’s take a 5min break and see if you can create your Slack account, start downloading Anaconda, if using, (and get some refreshments, take a bio-break etc)

Terminology

- Programming language provides set of primitive operations (primitives), e.g., “=“, “+”
- Primitives can be combined (with certain rules) into expressions, e.g. “a = b+x”
- Expressions manipulate data and can themselves have a value, e.g., “1+1 == 2” has Boolean value “True”
- Expressions can be typed in Python shell directly one-by-one or stored in a file and executed in sequence → program (“Python script”)

A Python script

- 

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Jul 10 15:47:37 2022

@author: rolandg
"""

# say hello (this is a comment)
print("Hello world!")

# that's the end of my script
#
```

this is the directory where my Python scripts are stored

- when running the script (e.g. in spyder):

```
In [183]: runfile('/Users/rolandg/Python/Hello.py', wdir='/Users/rolandg/Python')
Hello world!

In [184]:
```

Data

- Expressions can be used to manipulate data
- Data in Python are data objects
- Data objects have different types
- Type describes nature of data (e.g. string of characters or floating point number or complex data structure)
- Type determines the kind of operations possible on data structure
- Data objects can be scalar (no internal structure, e.g., an integer number) or non-scalar (internal structure, e.g., a list of integer numbers)

Simple data types

```
In [140]: i = 5

In [141]: type(i)
Out[141]: int

In [142]: x = 25.5

In [143]: type(x)
Out[143]: float

In [144]: a = "Gunther"

In [145]: type(a)
Out[145]: str

In [146]: f = True

In [147]: type(f)
Out[147]: bool
```

- Unlike e.g., C++ (and most other compiled languages) types in Python are usually not assigned explicitly, but determined by Python from context/data/ assignment
- usually convenient, but can lead to surprises

Types can be converted

```
In [148]: i = 100

In [149]: x = float(i)

In [150]: print(x)
100.0

In [151]: type(x)
Out[151]: float
```

Combine objects and operators to form expressions

```
In [152]: x = 100.0

In [153]: y = 10.0

In [154]: z = x/y

In [155]: print(z)
10.0
```

Algebraic, e.g., +,-,*,/

```
In [159]: y == z
Out[159]: True

In [160]: y == x
Out[160]: False

In [161]: y < x
Out[161]: True
```

Logic, e.g., ==, <, >

note difference between “=“ and “==“

Expressions (“x/z” or “y == x”) have value and type

Expressions

- Type of expression is usually determined from input object type

```
In [162]: i = 10 + 10

In [163]: type(i)
Out[163]: int

In [164]: x = 10.0 + 10.0

In [165]: type(x)
Out[165]: float
```

- Parenthesis can be used to fix order of computation

```
In [170]: x = 10.0 + 20.0/2.0

In [171]: print(x)
20.0

In [172]: x = (10.0 + 20.0)/2.0

In [173]: print(x)
15.0
```

Equal sign “=“

- Equal sign assigns value to a variable (object)

```
In [180]: e_approx = 2.718

In [181]: print(e_approx)
2.718

In [182]: log(e_approx)
Out[182]: 0.999896315728952
```

- assignment binds name (e_approx) to value (2.718) stored in computer memory
- can bind name to another value with a new assignment
  - at that point, old value can’t be accessed any more

Some basics for understanding Python code

Whitespace is meaningful in Python: especially **indentation** and placement of newlines

- Use a newline to end a line of code
  Use \ when must go to next line prematurely
- No braces {} (as in C++) to mark blocks of code, use *consistent* indentation instead
  - First line with *less* indentation is outside of the block
  - First line with *more* indentation starts a nested block
- Colons start of a new block in many constructs, e.g. function definitions

Some basics for understanding Python code

- **Indentation matters (unlike e.g., C++)**
  - Block structure indicated by indentation

```
x = 9.0
print("With indentation")
if(x > 10.0):
    y = 10
    print("Yeah")

print("Without indentation")
if(x > 10.0):
    y = 10
print("Yeah")
```

```
In [185]: runfile('/Users/rolandg/Python/Hello.py', wdir='/Users/rolandg/Python')
Hello world!
With indentation
Withouy indentation
Yeah
```

Some basics for understanding Python code

- First assignment to a variable (name) creates it
  - Variable types don't need to be declared
  - Python figures out the variable types automatically
- Assignment is = and comparison is ==
- For numbers + - * / % work as expected
  - For strings, + provides concatenation
- Logical operators are words: "and", "or", "not"
- The basic printing command is print

Operators do different things for different types

```
In [186]: i = 10

In [187]: j = 20

In [188]: print(i+j)
30

In [189]: a = "abc"

In [190]: b = "def"

In [191]: print(a+b)
abcdef
```

Variable assignment

- *Binding a variable* means setting a *name* to hold a *reference* to some *object or value* in memory
  - *Assignment creates references, not copies*
- Names in Python do not have an intrinsic type, objects have types
  - Python determines the type associated with named object automatically, based on what data is assigned to it
- You create a name the first time it appears on the left side of an assignment expression:

```
x = 3
```

- A reference is deleted via garbage collection after any names bound to it have passed out of scope

Naming conventions

- Names are **case sensitive** and cannot start with a number. They can contain letters, numbers, and underscores, e.g.,

```
bob   Bob   _bob   _2_bob_   bob_2   BoB
```

  - I do NOT recommend using upper/lower case to distinguish variables (e.g., bob vs Bob) - just asking for confusion

- There is a number of reserved words that you can't re-use or redefine as they are intrinsic Python commands:

```
and, assert, break, class, continue,
def, del, elif, else, except, exec,
finally, for, from, global, if, import,
in, is, lambda, not, or, pass, print,
raise, return, try, while
```

Naming conventions

- Recommended naming conventions:
- **joined_lower** for functions, methods and attributes
- **ALL_CAPS** for constants
- **StudlyCaps** for classes
- **camelCase** if needed to conform to pre-existing conventions
- Sticking to these conventions will make your code easier to read for others (and for yourself)
- For temporary variables I usually use i,j,k… for integers, x,y,z,.. for floats and a,b,c… for strings

Some options for assignments

- Make multiple assignments at the same time

```
In [18]: x,y = 4,8

In [19]: print(x,y)
4 8
```

Can swap variables very easily

```
In [20]: x,y = y,x

In [21]: print(x,y)
8 4
```

- Assignments can be chained (but not sure if I recommend overusing this)

```
In [22]: x = y = z = 123

In [23]: print(x,y,z)
123 123 123
```

- However, I suggest that for the beginning we write things as explicitly as we can - make it easy to read for non-Python experts

Control flow and branching

- For almost any program to be useful we need to control the flow of commands based on data and conditions we encounter → control flow, branching and iteration

Branching: “if”, “else”, “elif”

- Decide whether block of code should be executed based on a Boolean condition (“True” or “False”)

```
if <condition>:
    <expression>
    <expression>
    ...
```

```
if <condition>:
    <expression>
    <expression>
    ...
else:
    <expression>
    <expression>
    ...
```

```
if <condition>:
    <expression>
    <expression>
    ...
elif <condition>:
    <expression>
    <expression>
    ...
else:
    <expression>
    <expression>
    ...
```

- Recall importance of indentation; don’t forget “:”

Iteration: “while” loops

```
while <condition>:
    <expression>
    <expression>
    ...
```

- Enter loop if <condition> is “True”
- Continue executing expressions in loop repeatedly until condition becomes “False”
- If condition remains “True” will loop forever…

Iteration: “for” loops

- Iterate through a range of numbers

```
for <variable> in range(<some_num>):
    <expression>
    <expression>
    ...
```

- Start with variable in smallest value of range
- Execute block of code
- Increase variable by 1 and check if still in range
  - if True, execute block of code
  - if False, exit for loop
- Can do the same with a while loop, but needs more lines of code and easier to make mistakes

The “range” function in Python

- *range(start, stop, step)*
- Default values are *start = 0* and *step = 1*
- will count from *0* to *stop-1* in steps of 1
- can change start and stop values, as well as step size

“Sequence types”

- What if you have many numbers, letters etc that you want to access?
- → Sequence types: Tuples, strings and lists
- Can be “mutable” and “immutable”
- Lists and tuples can contain objects of different types
  - although in most cases we will use lists containing objects of uniform type

Examples

- **Tuple**: (‘Gunther’, ‘abc’, 123, [SomeOtherSequence])
  - An *immutable* sequence of items
  - Items can be of mixed types,
  - can include collection types
  - note the parenthesis
- **Strings**: “Gunther Roland”
  - *Immutable* sequence of characters
- **List**: [1, 2, 3, ’Gunther’, (‘left’, ‘right’)]
  - *Mutable* sequence of items of mixed types
  - note the square brackets

Sequence type syntax

- Tuples, strings and list share a lot of functionality and syntax (i.e., the way to access elements)
- Key difference:
  - lists are *mutable* (items in the list can be changed),
  - tuples and strings are immutable (e.g., cannot change the “o” in “Roland” to a different letter) - *immutable*
-

Sequence type creation - syntax

- Create tuples using parentheses and commas

```
In [24]: tu = (23,45,"abc","def",54)

In [25]: print(tu)
(23, 45, 'abc', 'def', 54)

In [26]: type(tu)
Out[26]: tuple
```

- Create lists are using square brackets and commas

```
In [27]: li = [23,45,"abc","def",54]

In [28]: print(li)
[23, 45, 'abc', 'def', 54]

In [29]: type(li)
Out[29]: list
```

- Create strings using quotes (“, ‘, or “““)

```
In [31]: st = "23,45,abc,def,54"

In [32]: print(st)
23,45,abc,def,54

In [33]: type(st)
Out[33]: str
```

Accessing member elements of sequences

- Use square bracket “array” notation

```
In [34]: print(tu[1])
45

In [35]: print(li[1])
45

In [36]: print(st[1])
3
```

Note that we are counting **from 0** - first element is accessed by li[0]

Interesting feature in Python (for a C++ programmer)

```
In [37]: print(li[0])
23

In [38]: print(li[-1])
54

In [39]: print(li[-5])
23
```

Positive index: count from “left”, starting with 0

Negative index: count from “right”, starting with –1

Important operation: “slicing”

- Return a sequence of same type, containing a copy of a subset of the original sequence

```
In [40]: li = [23,45,"abc","def",54]

In [41]: print(li[1:3])
[45, 'abc']

In [42]: type(li[1:3])
Out[42]: list
```

- Start copying at the first index
- Stop copying ***before*** second index
- Negative indices count from “right”

Copying the whole sequence

- [:] makes a *copy* of the entire sequence
- IMPORTANT: Note the difference between these two operations for lists:

```
In [43]: li = [23,45,"abc","def",54]

In [44]: li2 = li[:]

In [45]: li3 = li

In [46]: li[0] = 11

In [47]: print(li)
[11, 45, 'abc', 'def', 54]

In [48]: print(li2)
[23, 45, 'abc', 'def', 54]

In [49]: print(li3)
[11, 45, 'abc', 'def', 54]
```

Makes a copy

Points to same list

Useful operator: “in”

- Test whether a value is inside sequence (True or False):

```
In [50]: tu = (1,2,3,5,6)

In [51]: 1 in tu
Out[51]: True

In [52]: 2 in tu
Out[52]: True

In [53]: 4 in tu
Out[53]: False
```

- Tests whether substrings is inside string:

```
In [54]: st = "Gunther"

In [55]: "un" in st
Out[55]: True

In [56]: "e" in st
Out[56]: True

In [57]: "a" in st
Out[57]: False
```

Useful operator: “+”

The “+” operator produces a *new* sequence by concatenation of arguments:

```
In [58]: (2,4,6)+(1,2,3)
Out[58]: (2, 4, 6, 1, 2, 3)

In [59]: "Gunther" + "Roland"
Out[59]: 'GuntherRoland'
```

Somewhat useful operator: “*”

The “*” operator produces a *new* sequence by concatenating arguments *n* times:

```
In [60]:  "Gunther" * 4
Out[60]: 'GuntherGuntherGuntherGunther'

In [61]: (1,2,3) * 4
Out[61]: (1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3)
```

“Mutability” - lists are mutable

```
In [64]: li = [1,2,3,4,5]

In [65]: li[4] = li[0]

In [66]: print(li)
[1, 2, 3, 4, 1]
```

- Changes list *in place* - change content of memory list is pointing to

“Mutability” - tuples are NOT mutable

```
In [67]: tu = (1,2,3,4,5)

In [68]: tu[4] = tu[0]
Traceback (most recent call last):

  File "<ipython-input-68-f2c37bc18289>", line 1, in <module>
    tu[4] = tu[0]

TypeError: 'tuple' object does not support item assignment
```

- Cannot change a tuple in place
- If you want a different value in tuple, need to make new one
- Why tuples? Immutability means operations are faster on tuples than on lists

Important operators for lists: “append” and “insert"

Lists are mutable - can be extended:

```
>>> li = [1, 11, 3, 4, 5]

>>> li.append('a')  # Note the method
  syntax
>>> li
[1, 11, 3, 4, 5, 'a']

>>> li.insert(2, 'i')
>>>li
[1, 11, 'i', 3, 4, 5, 'a']
```

Other powerful built-in functions for lists

Lists have many methods, including index, count, remove, reverse, sort

```
>>> li = ['a', 'b', 'c', 'b']
>>> li.index('b')  # index of 1st occurrence
1
>>> li.count('b')  # number of occurrences
2
>>> li.remove('b') # remove 1st occurrence
>>> li
  ['a', 'c', 'b']
```

Other powerful built-in functions for lists

```
>>> li = [5, 2, 6, 8]

>>> li.reverse()    # reverse the list *in place*
>>> li
  [8, 6, 2, 5]

>>> li.sort()       # sort the list *in place*
>>> li
  [2, 5, 6, 8]

>>> li.sort(some_function)
    # sort in place using user-defined comparison
```

More on tuples

- The **comma** is the tuple creation operator, not parenthesis

```
>>> 1,
(1,)
```

- Parenthesis is just shown for clarity

```
>>> (1,)
(1,)
```

- No comma, no tuple

```
>>> (1)
1
```

- Empty tuples have a special syntactic form

```
>>> ()
()
>>> tuple()
()
```

tuples vs lists

- Lists slower but more powerful than tuples
  - Lists can be modified, and they have lots of handy operations and mehtods
  - Tuples are immutable and have fewer features
- Can convert between tuples and lists using the list() and tuple() functions:

```
li = list(tu)
tu = tuple(li)
```

- I find myself using lists more often - can do anything tuples can

Input

- Often (always?) need to get input data from user or from file on disk
- Asking for user input: “input()” (what else?)

```
In [7]: a = input("enter a name:")

enter a name:Gunther

In [8]: print(a)
Gunther

In [9]:
```

Input from text file

- We will mostly use text files as data source
- Numbers or text entries, separated by “,” (“CSV”), whitespace “ “ or some other delineator
- e.g.:

```
14.Sep.B01.01,1,9,0.2,10.4,0.2,36.8,0.3,3400,200
14.Sep.B01.02,2,9.1,0.6,10.5,1,36.8,0.8,3500,400
14.Sep.B01.03,3,8.9,0.1,10.1,0.3,36.5,0.3,3281,0.44
14.Sep.B01.04,4,9,0.5,10.4,0.5,36.9,0.5,3400,0.9
14.Sep.B01.05,5,8.9,0.1,10.4,0.1,37,0.1,3400,51
14.Sep.B01.06,6,9,0.3,10.4,0.3,36.7,0.5,3435,158.5
14.Sep.B01.07,7,9.1,0.2,10.5,0.2,36.9,0.2,3510,100
```

- There are many ways to do this in Python. We’ll start with the simplest (?) and use other methods later

Input from text file

- s = f.readline() reads one row from a data file as a single string, here assigned to “s”
- can use t = split(“,”) to break the string into smaller strings separated by the delineator (here comma, “,”), t will be of type “list”
- e.g.:

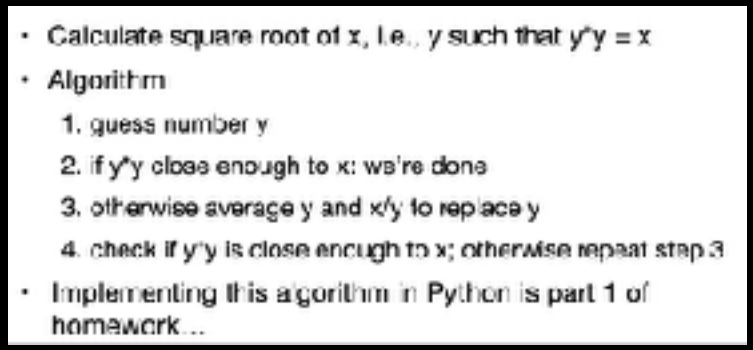
```
In [9]: f = open("/Users/rolandg/Python/cubeData.csv","r")

In [10]: s = f.readline()

In [11]: print(s)
14.Sep.B01.01,1,9,0.2,10.4,0.2,36.8,0.3,3400,200


In [12]: t = s.split(",")

In [13]: print(t[0])
14.Sep.B01.01

In [14]: print(t[9])
200
```

Homework - I

- If not done yet: Create free account on Slack and join Slack channel
- If not done yet: Install Python framework (e.g., Jupyter) on your laptop
- Send photo with chinese and english name to TA

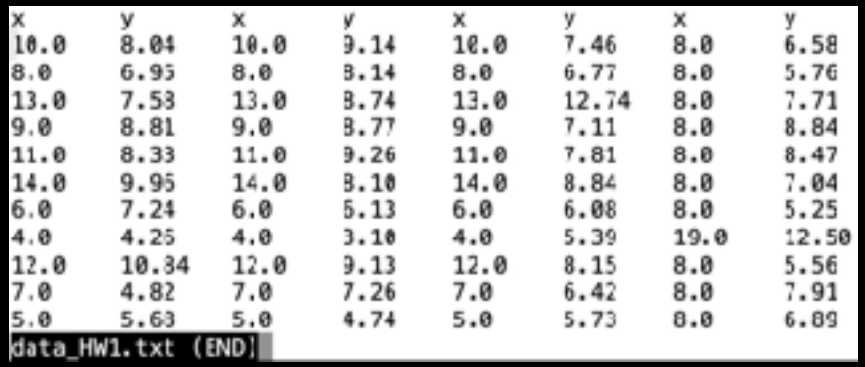
Homework - II

- Write a short Python script that asks the user for a number and finds the square root of the number using the algorithm on the right
  - do not use any math operators other than +,-,*,/
- Send your working Python code to TA
- You can collaborate, but every student needs to write and submit their own code!



Homework - III

- Write a short Python script that reads the file “data_HW1.txt”. The file has 8 columns. Calculate the average of the numbers in each column in a for loop and use numpy.std() to calculate the standard deviation of each column (we’ll discuss later how that is defined).
- Send your working Python code to TA
- You can collaborate, but every student needs to write and submit their own code!



x	y	x	y	x	y	x	y
10.0	8.04	10.0	9.14	10.0	7.46	8.0	6.58
8.0	6.95	8.0	8.14	8.0	6.77	8.0	5.76
13.0	7.58	13.0	8.74	13.0	12.74	8.0	7.71
9.0	8.81	9.0	8.77	9.0	7.11	8.0	8.84
11.0	8.33	11.0	9.26	11.0	7.81	8.0	8.47
14.0	9.96	14.0	8.10	14.0	8.84	8.0	7.04
6.0	7.24	6.0	6.13	6.0	6.08	8.0	5.25
4.0	4.26	4.0	3.10	4.0	5.39	19.0	12.50
12.0	10.84	12.0	9.13	12.0	8.15	8.0	5.56
7.0	4.82	7.0	7.26	7.0	6.42	8.0	7.91
5.0	5.68	5.0	4.74	5.0	5.73	8.0	6.89

data_HW1.txt (END)

```
In [16]: x = [1,2,3,4,5,6,7,8,9]

In [17]: np.std(x)
Out[17]: 2.581988897471611
```